APPLICATION FOR UNITED STATES LETTERS PATENT

For

MULTIPLE-IMAGE VIEWER

Inventor:

James A. Munro

Prepared by:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
32400 Wilshire Boulevard
Los Angeles, CA 90025-1026
(408) 720-8300

Attorney's Docket No.: 03971.P014

MULTIPLE-IMAGE VIEWER

RELATED APPLICATIONS

[001] This application claims the benefit of U.S. provisional Application No. 60/175,303, filed January 10, 2000.

FIELD OF THE INVENTION

[002] The present invention relates to the field of image display; more particularly, the present invention relates to displaying multiple images that are independently manipulatable in a single window.

NOTICE OF COPYRIGHT PROTECTION

[003] Incorporated herein is a version of an extended markup language manual detailing how to use this language. The manual is the PIXML specification 1.0, Copyright, 2000, PicSurf Inc. A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the " PIXML specification 1.0", as it appears in the Patent and Trademark Office Patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

[004] Today, many people access images from the Internet for a variety reasons. For example, when shopping over the Internet, shoppers have two desires. Shoppers want to see images of what they are buying. Shoppers want to see these images in real time.

[005] When images are being displayed using a web browser, a page description language, such as, for example eXtensible Markup Language (XML) or Hyper Text Markup Language (HTML), defines how to display these images. The standard HTML language allows images of various types, such as, for example, raster graphics and vector graphics, to be inserted into a web page using the HTML tag 'IMG'. Images from raster graphic files do compress but generally not efficiently with current compression technologies. Traditionally, access to view such images over the web is slow or takes up considerable bandwidth. Traditionally, shoppers do not generally wait for slow web pages to download.

[006] Shoppers also like to compare and manipulate images of products that they are considering buying. Flashpix is one implementation that allows people to view and manipulate an image by zooming in on the image. Live Picture and others use Internet Imaging Protocol (IIP) to manipulate an image in the window. However, these implementations are not useful for manipulation of multiple images displayed in a single window. Currently, an Internet shopper is inhibited from displaying and manipulating multiple images of competing products in a single window.

[007] In most client-server systems for viewing or browsing different types of content, when an image is displayed, the underlying system reserves a rectangular area on the

screen in which the image is displayed. In the case of a typical web page written in HTML, when the browser encounters an image, the browser creates a window in which the image is displayed. Usually, the browser creates the window, and then the image software decodes and displays the image into the window. In the case of no-standard, or non embedded image types, the browser software creates a window for the image, and then passes control to a software 'plug-in' which decodes and displays the image in the window. Most web browsers can display images directly even when they are not embedded in an HTML or other document. In this case, the browser allocates the entire browser window as the image window. However, whether the viewing application is a web browser, or a Java application (applet), the concept of an image 'window' is universal and each allocated window space is generally occupied by a single image.

[008] Today, images may be displayed over a network within a window in a variety of ways. Multiple images with each image in its own window, usually an array of thumbnails, may be displayed together within a single overall browser window. A composition of multiple images may be put together into a single image file and that sole image file will be displayed within a single window. In video applications, a display of a sequence of multiple images occurring one at a time takes place in a single window. Yet none of these applications allow for two separate images, each image having an independent data file, to be concurrently displayed and manipulated in the same window.

SUMMARY OF THE INVENTION

[009] A method, apparatus, and system in which a multiple-image viewer concurrently displays and manipulates multiple images within a single window in a network system. One or more of the displayed images are a raster graphic file. Each of the displayed images has a separate data file.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The drawings refer to the invention in which:

[0011] figure 1 illustrates an embodiment of a client-server system using the multiple-image viewer;

[0012] figure 2 illustrates an exemplary web page using an embodiment of the multiple-image viewer to display four images and the content associated with those images;

[0013] figure 3 illustrates the third image of figure 2 magnified by an embodiment of the multiple-image viewer;

[0014] figure 4 illustrates the independent nature of each image file within the window;

[0015] figure 5 illustrates that the entirety of the window space is available for any images displayed within that window and that the images may overlay one another.;

[0016] figure 6 illustrates the first through fourth image as shown in figure 2; however, the user has selected a region of interest and zoomed in on a majority of the third image and small portions of the first image, the second image and the fourth image;

[0017] figure 7 illustrates an embodiment of the multiple-image viewer displaying a wine bottle image and a hierarchical system of folders containing content associated with that wine bottle image;

[0018] figure 8 illustrates the corresponding size of the data file associated with each level of resolution of a displayed image; and

[0019] figure 9 illustrates an embodiment of a multiple-image viewer implemented as a program containing various modules.

[0020] While the invention is subject to various modifications and alternative forms, specific embodiments thereof have been shown by way of example in the drawings and will herein be described in detail. The invention should be understood to not be limited to the particular forms disclosed, but on the contrary, the intention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention.

DETAILED DISCUSSION

[0021] A multiple-image viewer is described. In the following description, numerous details are set forth, such as specific controls to manipulate an image, specific methods to calculate a predetermined setting, etc. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

[0022] Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as data bits, values, elements, symbols, characters, terms, numbers, or the like.

[0023] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions

utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0024] The multiple-image viewer also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[0025] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the description below. In addition, the multiple-image viewer is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

Overview

[0026] In an embodiment, a multiple-image viewer system is described that may display and/or manipulate multiple images in a single window, such as a browser window or plug-in window. In an embodiment, the window of the multiple-image viewer may be the viewing area or display area reserved for the purpose of displaying one or more images and any content associated with those images. The top-level of the window may be the plug-in display window. Image file and compression technology supports the multiple-image viewer. Although at least one image file and compression technology are described herein, it would be apparent to those skilled in the art to employ other image file structures and/or different compression technologies.

[0027] Figure 1 illustrates an embodiment of a client-server system **100** using the multiple-image viewer **102**. The client-server system comprises a client **104** having a cache **106** and an embodiment of the multiple-image viewer **102**, a network connection **108**, a server **110**, and an image database **112** associated with the server **110**. In an embodiment, the client **104** may be a personal computer or other similar device. In an embodiment, the network connection **108** may be a digital subscriber line connection, a T-1 connection, a local area network connection, an Internet server provider connection, wireless connection, or other similar network connection. In an embodiment, a network may be a client server system, a World Wide Web, an Internet, a mobile phone network, a first device in communication with a second device, such as a computer in communication with a first personal digital assistant (PDA), a first PDA in

communication with a second PDA, or a PDA in communication with an intelligent phone, or any other similar system. In an embodiment, the multiple-image viewer **102** instructs the client **104** to request image data **114** from the image database **112** via the server **110**. In one embodiment, multiple-image viewer **102** displays and enables manipulation of multiple images through the use of a web-based application. The multiple-image viewer **102** may be integrated into a browser, a plug-in, an Active-x control, a Java applet, or another similar program. The browser may be a readily available Internet web browser software product (e.g., a browser available from Netscape, Internet Explorer, a Java-implemented browser, etc.). In an alternate embodiment, the browser may be implemented as a stand-alone Java applet or an Active-X control. In one embodiment, the browser allows functionality to be extended by plug-ins. Thus, the plug-in extends the browser's architecture to allow the images to be displayed in the window.

[0028] In one embodiment, the image database **112** associated with the server **110** stores images. Each of the images may have a separate image data file **114**. The image data file **114** may be stored in a compressed format. In one embodiment, the image data file **114** is compressed according to a block-based integer wavelet transform entropy-coding scheme.

[0029] In one embodiment, the multiple-image viewer **102** uses the standard HTML language to insert images into web pages. In one embodiment, in this case, the images are inserted into the window using the HTML 'EMBED' tag. That is, in one embodiment, the HTML syntax is an extension to the existing EMBED tag. Using features of the HTML language, the size and position of the image window can be controlled. In an embodiment, the viewer uses another Extended Markup Language, PIXML, to insert

images into a web page. An embodiment of the PIXML is attached to this description and incorporated herein. In another embodiment, the multiple-image viewer may use XML or another similar page description language to insert images into a page.

[0030] In an embodiment, the browser creates a window for the multiple-image viewer **102** and obtains the data associated with the multiple-image viewer **102**. The browser then relinquishes control to the multiple-image viewer **102**, which decodes image data **114** and displays the images in the window. Thus, the multiple-image viewer **102** decodes and displays multiple images within a single plug-in window.

[0031] By extending the functionality of a plug-in window in this way, the multiple-image viewer **102** is able to perform operations on these images, either as a group or as individuals. For example, the user can visually zoom in or out on a group of images. In one embodiment, to facilitate these operations, each image may have its own specific hypertext links, image map, or other attributes to allow the images to be manipulated independently.

[0032] Figure 2 illustrates a web page **200** using an embodiment of the multiple-image viewer to display four images and the content associated with those images. In one embodiment, a browser displays the web page **200**. The web page **200** or page file contains two text documents **204**, an index **206** with hypertext links, and a window **202** created by an embodiment of the multiple-image viewer. The window **202** contains four images; first image **208**, a second image **210**, a third image **212**, and a fourth image **214**. The window also has two navigation tool bars **216** with numerous controls **230** to manipulate these images. The first image **208** displayed by the viewer is a digital image of an oil painting of a sun over two smiley faces. The second image **210** is of a knight

riding a horse through a wooded countryside. The third image **212** is a digital photograph of a woman talking on the phone. The fourth image **214** is a bottle of wine. In one embodiment, the navigation tool bars **216** contain the following controls **230** to manipulate each image. The user may manipulate each of the displayed multiple images by zooming in on the image, zooming out from the image, selecting a region of interest in the image, restoring the default or initial view of the image, panning the image, linking to the image, stretching the entire image, centering the image in the window, resetting/undoing the last operation performed on the image, magnifying the image, moving left on the image, moving right on the image, moving up on the image, or moving down on the image. In another embodiment, the multiple-image viewer may manipulate a displayed image by using the controls **230** mentioned above as well as other similar controls.

[0033] The manipulation controls **230** from the navigation tool bar **216** may be implemented in the client. As noted above, the user can use the set of controls **230** to zoom in/out or to pan across the images. By zooming or panning, the user causes the multiple-image viewer to calculate new parameters and then make the appropriate request for data (e.g., blocks of data) to the server. When the user zooms in on the images, the multiple-image viewer calculates the new geometric coordinates for the new view. Based on the location of the cursor, the multiple-image viewer calculates which part of the image(s) will appear in the window **202** and then obtains the appropriate data. Based on this determination, the client makes a simple request to the server and the server responds with the appropriate block(s) of data. Using the data, the multiple-image viewer calculates where in the window **202** each part of each image is to appear. For multiple

images, this process is repeated for each image in the window **202**. Note, the images are shown in a regular array, i.e., the images are evenly spaced between each other and arranged in a linear manner. The images in the window **202** may also be located as an irregular array to allow each image to have different size dimension and even overlay on top of another image.

[0034] Figure 3 illustrates the third image of figure 2 magnified by an embodiment of the multiple-image viewer. A user has employed a control **320** of the multiple-image viewer to magnify the third image **312** displayed in the window **302**. The third image **312** has increased in viewing area to occupy the entire window **302**.

[0035] Figure 4 illustrates the independent nature of each image file within this window. Each displayed image has a separate and independent data file. Thus, an end user may manipulate each image independently in almost any manner the end user chooses. The first image **402** of the sun and the smiley faces has been stretched. Thus, the displayed sun and the displayed smiley faces are slightly distorted in the lateral direction. The second image **404** has been zoomed-in **406** on. Thus, the display sizes of the knight and the countryside have increased within the window **401**. The third image **408** has been condensed and moved to occupy a partial amount of the space where the fourth image **410** was located. The fourth image **410** has been condensed and moved to occupy a partial amount of the space where the third image **408** was located.

[0036] Figure 5 illustrates that the entirety of the window space is available for any images displayed within that window and that the images may overlay one another. The first image **504** has been manipulated to be increased in display size within the window **502**. The display of the first image **504** now overlays portions of the second image **506**,

the third image **508**, and the fourth image **510**. The window **502** is an area reserved to display one or more images. Any one of the displayed images may occupy part of the window **502** or the entirety of the window **502**

[0037] Figure 6 illustrates the first through fourth image as shown in figure 2; however, the user has selected a region of interest and zoomed in on a majority of the third image and small portions of the first image, the second image and the fourth image. The viewer displays a majority of the third image **612**, a woman speaking on a telephone, and only small portions of the first image **608**, the second image **610** and the fourth image **614**.

[0038] In one embodiment, the multiple-image viewer constantly keeps track of which data it already has so that it does not have to request the same data multiple times from the server. In one embodiment, the multiple-image viewer keeps track of what is in the window and also what other data is in the cache. A pixel-to-pixel mapping exists between the image and the window, so depending on resolution level, window size, and image position within (or without) the window, the client performs the geometric calculations.

[0039] In one embodiment, in the case of zooming, panning, or moving, when the proper data to fill in a new part of an image displayed in the window is not available, then the data is scaled from the previous resolution level and used immediately. When the proper data arrives from the server, the data is decoded and displayed. Thus, when the user moves, pans, or zooms, an immediate visual result occurs with the quality of the image improving as data arrives.

[0040] In one embodiment, the request for data is performed using a HTTP 'GET' command that specifies the URL of each image, which resolution level, and which blocks

of data are required based on, for example, resolution level. In an embodiment, the default is to obtain the entire full size image. In one embodiment, the multiple-image viewer only requests image data for those images or parts of images, which actually appear in the plug-in window. Images or parts of images that are outside the visible plug-in window are not requested to preserve bandwidth. Note if the multiple-image viewer requests data for two or more of the images, then the image date files may be on different sites.

[0041] In one embodiment, all data received from the server is cached locally and reused wherever possible. Caching data locally allows random access to different parts of the image and allows images, or parts of images, to be loaded in a variety of resolution and quality levels. In one embodiment, the multiple-image viewer reuses the existing image data together with the new image data to create a high quality higher resolution view. Thus, the multiple-image viewer uses a file hierarchy that allows for two resolution levels to be extracted from one sub-image. In an alternative embodiment, the client initially downloads all the images. At which point, the multiple-image viewer only decodes that portion of each image that is to appear in the window. In an embodiment, the multiple-image viewer requests and decodes the amount of data corresponding to an actual area of the image to be displayed, blocks of data surrounding that area to be displayed, and data for one level of higher resolution of the image being displayed.

[0042] Figure 7 illustrates an embodiment of the multiple-image viewer displaying a wine bottle image **702** and a hierarchical system of folders **704** containing content associated with that wine bottle image **702**. The window **706** displays the wine bottle image **702** and four icons, a German wine icon **708**, a French wine icon **710**, an Italian

wine icon **714**, and a California wine icon **712**. In one embodiment, a hierarchical folder such as a parent folder contains the image of the wine bottle **702** and four subfolders **708**, **710**, **712**, **714**, represented by the icons. In another embodiment, the image of the wine bottle **702** is separate from each folder represented by an icon and each folder represented by an icon is distinct from every other folder. The author of the web page may determine how to arrange these items that appear in the window **706**, such as images, folders, and content within either the image or the folder. However to the user, the window **706** appears to be displaying the same picture.

[0043] The multiple-image viewer allows for images to be comprised of a hierarchical system of folders **704**. The multiple-image viewer uses two basic objects an image and a folder. A folder is a container that can hold, and thus display, one or more images. An image may be a raster graphic (i.e. natural bitmap image) or other similar file. A raster graphic differs from a vector graphics in the way that a computer interprets the image data file. A vector graphic defines a picture as points, lines and other geometric entities. The points, lines and other geometric entities generally define an object. The combination of all the individual objects usually creates the vector graphics image. A raster graphic represents a picture image as a matrix of dots known as pixels. The computer generally views the combination of all of the pixels to comprise the image. Dozens of raster (natural bitmapped) graphics formats exist, including GIF, TIF, BMP, JPG and PCX. The image may be encoded with compression technology and with multi-resolution random access capability. Both a folder and an image can have other content associated with them. Both images and folders can contain content such as images, graphics objects, sub folders, tiled and non-tiled background images, a text document, a hyperlink, an

image map, an image address or other similar content. In an embodiment, each folder may be represented in XML by a <PIXML> tag.

[0044] Both images and folders have a variety of attributes that include a flexible way of defining behaviors such as zooming or moving objects. The multi image viewer also supports an event manager that enables external user code to respond to events that occur within the system. Graphic objects include basic 2-D vector graphics functions such as text, lines, and circles. Images can be placed in separate layers; the upper layer will overlay the lower one when there is an overlap. Each image can also have a hypertext link so that the user can click on a specific image and cause the browser to go to a new location in the image. In addition to being able to display a folder as an image, a web page author may also use an icon, thumbnail, or other similar structure to visually represent the folder.

[0045] In one embodiment, the multiple-image viewer displays an icon representing either an image, a folder, the content within the folder, or the content within the image, if the level of the image is below the value of a predetermined setting. Similarly, the multiple-image viewer displays the image, the folder, the content itself, if the level of the image is above the value of a predetermined setting. Thus in this illustration, if a user zooms in on the folder represented by a French wine icon **710**, then a French wine subfolder **720** opens up to reveal four more subfolders, a year 1991 subfolder **722**, a year 1992 subfolder **724**, a year 1993 subfolder **726**, and a year 1994 subfolder **728**. If the user zooms in on the icon representing the 1992 subfolder **724**, then another subfolder **726** opens up to reveal numerous wine bottle icons **730** labeled with the types and manufacturer of the actual 1992 wines that the user may purchase, for example. If the

user enlarges a specific wine bottle icon **730** above the predetermined setting, then a full image of the specific wine will be displayed in the window **706**. In another embodiment, the content will be hidden within the image if the level of the images is below a predetermined setting. As described herein, in an embodiment the multiple-image viewer only downloads the data for the images, folders, and subfolders actually displayed. Thus, when the user requests another subfolder to open up, then the viewer downloads those corresponding blocks of data.

[0046] The multiple-image viewer may use a variety of methods to establish the value of the predetermined setting for displaying or not displaying an image, an icon, or content. The predetermined setting may be selected from one of the following: a level of zoom, a predetermined resolution level, a preset size of the image or folder to the viewing area, a percentage of the full sized original image, a display level, or a similar mathematical arrangement. In an embodiment, when an object, an image or a folder is displayed, the level of the object is combined with the level of the parent folder, and the levels of all the parent folders to compute a display level for display purposes. The viewer may compare the display level to a root level to determine whether or not the value is above or below the value for the predetermined setting. In an embodiment, the author of the web page determines the value for the predetermined setting. In general, "zoom in" will reduce the level of the root folder by 1 and "zoom out" will increase the root folder level by 1. Zoom in and zoom out functions can also be defined by attribute values that can be integer resolution levels, percentages, or 'fit', where the zoom operation matches the resolution of the object(s) to the parent folder.

[0047] In an embodiment, the value of the predetermined setting is assigned to a "display level" attribute. The "display level" attribute is used to determine whether a folder is displayed as an image, a folder icon, not displayed, or whether the contents of the folder or image are contain within the object. In an embodiment, going above the predetermined setting causes a client to request more data, such as the entire data file for the image and any content within the image, from the server.

[0048] In an embodiment, below the value of the predetermined setting, the client downloads a limited amount of data regarding the content within the image and above the value of the predetermined setting; the client downloads the entire data file for the content. In an embodiment, the viewer requests and decodes the amount of data pertaining to the actual area of the image to be displayed, blocks of data surrounding that area to be displayed, and one level of higher resolution of image being displayed.

[0049] Figure 8 illustrates the corresponding size of the data file associated with each level of resolution of a displayed image. The multiple-image viewer supports displaying images having multiple levels of resolution. In an embodiment, an image may have four levels of resolution. The thumbnail image or icon image **802** has the lowest resolution level and the least amount of data in its corresponding image data file. The second resolution level **804** and the second highest resolution level **806** each have a progressively higher resolution level for the image and a greater amount of data in their corresponding image data file. The fourth resolution level or full size image **808** resolution is the highest resolution level and contains the greatest amount of data. If for example the user zooms in on an image above the predetermined setting, then the multiple-image viewer would request the next higher resolution level of the image. The multiple-image viewer

also allows arranging multiple images and graphics at different resolution levels, in the same window. An embodiment of a file structure along with multi-resolution compressed image management is described in U.S. Patent No.6,041,143, entitled "Multiresolution Compressed Image Management System and Method" issued March 21, 2000. An embodiment for transforming, quantizing, encoding, and/or building a resolution hierarchy which enables efficient coding at all levels of resolutions is described in U.S. Patent application No. 09/687,467, entitled "Multiresolution Image Data Management System and Method based on tiled wavelet_like transform and sparse data encoding," filed October 12, 2000 assigned to the corporate assignee of the present invention and incorporated herein by reference.

[0050] The compressed images are stored in a file structure. In one embodiment, the file structure comprises of a series of sub-images, each one being a predetermined portion of the size of its predecessor (e.g., 1/16 of the size of its predecessor). In one embodiment, each sub-picture is made up of a series of blocks that each contains the data associated with a 64 x 64 pixel block. That is, each image is divided into smaller individual blocks, which are 64 x 64 pixels. Each block contains data for decoding the 64 x 64 block and information that can be used for extracting the data for a smaller 32x32 block. Accordingly, each sub-image contains two separate resolutions. When the image is compressed, the bit-stream is organized around these 64 x 64 blocks and server software extracts a variety of resolution and/or quality levels from each of these blocks. The viewer stores in the cache the blocks of data for the image and areas substantially surrounding the displayed area

[0051] The server sends the client a portion of the file that includes parameters that detail image size (e.g., height and width), size of window resolution level, which blocks to decode, and the number of sub-pictures that are contained in the file. Initially, the images that are displayed in the window are set by the HTML tags or, in their absence, by default values.

[0052] In one embodiment, when the browser hands over control to the client side plug-in, the multiple-image viewer receives a set of parameters associated with the EMBED tag. These parameters include a list of image addresses, together with a set of parameters for each image that include image size, initial resolution level, and whether the image has a border. The plug-in parameters can specify which part of an image to load by defining a rectangular set of blocks. The default is the entire image. The plug-in makes the appropriate requests for data from the server side using standard HTTP protocols and then displays the set of images within the window. The multiple-image viewer automatically determines which blocks are within the window and only requests and decodes those blocks of data.

[0053] As noted above, in one embodiment, the images are compressed according to a block-based integer wavelet transform entropy coding scheme. For more information on one embodiment of the transform, see U.S. Patent No. 5,909,518, entitled "System and Method for Performing Wavelet-Like and Inverse Wavelet-Like Transformation of Digital Data," issued June 1, 1999. One embodiment of a block-based transform is described in U.S. Application Serial No. 60/094,129, entitled "Memory Saving Wavelet-Like Image Transform System and Method for Digital Camera and Other Memory Conservative Applications," filed July 22, 1999. One embodiment of scalable coding is

described in U.S. Patent Application No. 5,949,911, entitled "System and Method for Scalable Coding of Sparse Data Sets," issued September 7, 1999. One embodiment of block based coding is described in U.S. Patent No. 5,886,651, entitled "System and Method for Nested Split Coding of Sparse Data Sets," issued March 23, 1999. Each of these are assigned to the corporate assignee of the present invention and incorporated herein by reference.

[0054] Figure 9 illustrates an embodiment of a multiple-image viewer implemented as a program containing various modules. In an embodiment, the multiple-image viewer comprises a web-based program **900** consisting of the following modules to perform all of the functions previously described herein. A first module **902** exists to create a window defined by a page description language. A second module **904** exists to calculate the data to appear in the window and to request from a server data to appear in the window. A third module **906** exists to determine the value for the predetermined setting. A fourth module **908** exists to decode and display multiple images within the window. A fifth module **910** exists to display one or more images having a hierarchical structure and/or one or more folders having a hierarchical structure. A sixth module **912** exists to display one or more images having multiple levels of resolution. A seventh module **914** to display and manipulate one or more images compressed according to a block based integer wavelet transform entropy coding scheme. An eighth module **916** exists to enable controls for the manipulation of one or more images. A ninth module **918** to scale a displayed image to new size. A tenth module **920** to track the data being displayed in the window and to track what data is currently stored locally in the cache. In an embodiment, a computer program or another similar program directs and controls the operation of the

multiple-image viewer. The computer program is comprised of a number of modules to perform all of the functions previously described herein.

[0055] An embodiment of a multiple-image viewer, implemented as a program, can be embodied onto a machine-readable medium. A machine-readable medium includes any mechanism that provides (e.g., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

[0056] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims which in themselves recite only those features regarded as essential to the invention.

[0057]

Appendix

PIXML Specification 1.0

Abstract

This specification defines the features and syntax for the PicSurf Middleware Language, a language for describing and arranging multiple images and graphics at different resolution levels in XML.

PIXML is written in XML.

The system has been redesigned as a hierarchical system of folders. Each folder is represented in XML by a <pixml> tag. Henceforth in this document, the terms "<pixml>" and "folder" shall be used interchangeably. A folder can contain images, graphics objects, and daughter folders. Tiled and non-tiled background images are also supported. Graphics objects are attached to either folders or to images.
Each type of object has a variety of attributes that include a flexible way of defining behaviors such as zooming or moving objects. The system also supports an event manager that enables external user code to respond to events that occur within the system.

Graphics objects include basic 2-D vector graphics functions such as text, lines, and circles. Both PicSurf images and vector graphics can be placed in separate layers; the upper layer will overlay the lower one when there is an overlap.

The 2-D graphics functions are a subset of SVG, the W3C standard for scalable vector graphics. We expect that PIXML will eventually fully integrate with SVG.

The new system also includes behavior-definitions for many types of object and with which you can specify how items zoom and pan at different resolution layers.

PIXML is based on object-based reusable files. In PIXML, any element between <"TAG"> and </"TAG"> can be treated as an object. An element ID attribute allows it to be re-used. External code, usually Java Script or VB Script can specify any attribute of the element when it is used.

In some cases, you might manipulate more than one element. To avoid name conflict, you might refer it as (URL) "id".

PIXML has been designed with features that facilitate highly dynamic applications that include a high level of user interaction. The following are some of these features:

PIXML code can be dynamically generated
PIXML objects are reusable and attributes can be dynamically specified.
Client-side scripts can manipulate object attributes
Client side scripts can be triggered by events that occur within PXML
PXML can dynamically request new PXML scripts from the server
Dynamic manipulation of images and graphics is facilitated by specific implementation of important real-time functions such as smooth zoom in/out, panning, move, and drag.

Other new features include:

[0058] Background image offset feature for seamless integration with HTML.
[0059] Alpha blending and color transparency.

PIXML Introduction

PIXML stands for "PicSurf Images in XML. It is the XML-based scripting language that is used in PicSurf middleware.

PicSurf is implemented in various forms such as a browser plug-in, a Java applet, an Active-X control, or can be integrated with a browser. In all cases, PicSurf uses the PIXML scripting language to describe how PicSurf images and other components are arranged and displayed in a web browser, and how they behave under user interaction or under external software control such as JavaScript.

Why XML?

XML (Extensible Markup Language) deigned for documents containing structured information. PIXML has a very strong information structure that organizes the content (images and 2D graphics objects) into a tree and indicates what role the content plays (attributes). It is very natural to define PIXML in XML.

XML is a W3C standard. By following this standard we will attract more support and applications can be developed with less effort.
It also reduces the learning curve for those who want to add PicSurf features to their content.

XML is human readable and therefore has no exchange problems on Internet.

PIXML will be defined according to XML specification 1.0

PIXML Overview

Hierarchical Image Management

In PIXML, a tree-like hierarchical architecture is used to organize and manage images over the Internet. There are two basic objects used in PIXML: an image and a folder. An image is a raster graphic (i.e. natural bitmap image) encoded with PicSurf compression technology with multi-resolution random access capability.
A folder is a container that holds a set of images. The folder can in-turn contain sub-folders.
These two primary objects can have other objects associated with them. These include background images, 2-D graphics, and alpha-blending.

PIXML Structure

PIXML defines its own tags and the relationships between them in XML according to DTDs (Document Type Definitions).

In this chapter, we will go over each tag (element). In general, it starts from top to bottom order (from the whole file structure to separate elements, from parent tag to child tags). We also discuss each tag function and its definition.

First, we discuss the concept of XML Document Type Definitions (DTD). If you are familiar with DTD please skip over it.

DTD stands for "Document Type Definition" which deals with various types of declarations that are allowed in XML. More generally, declarations allow a document to communicate meta-information to the parser about its content. Meta-information includes the allowed sequence and nest of tags, attribute values and their types and defaults.

There are four kinds of declarations in XML: element, attribute list, entity and notation. The important declarations are element declarations and attribute list declarations.

Element declarations identify the names of elements and the nature of their content. A typical element declaration looks like this:

<!ELEMENT folder (Image+, bgImg, folder?)>

In this DTD element declaration, the element is of type 'folder' and it must contain one or more Image, exactly one bgImg and zero or more child folders.

[0060] The (+) after Image indicates that it may be repeated more than once but must occur at least once.

[0061] The question mark after folder(?) indicates that it is optional.

[0062] A name with no 'optional character' appended to it, such as bgImg, must occur exactly once.

Attribute declarations identify which element may have attributes, what attributes they may have, what values the attributes it may hold, and what default value each attribute has. A typical attribute declaration looks like this:

```
<!ATTLIST Image
id          ID          #implied
url         CDATA       #required
top         CDATA       "0"
left        CDATA       "0"
right       CDATA       "max"
bottom      CDATA       "max"
Level       CDATA       "0"
reset       (true, false)  "true"
>
```

Each attribute in a declaration has three parts: a name, a type and a default value.

There are 6 possible attribute types:

CDATA	String
ID	The value of ID attributes must be a name
IDRef or IDRefs	(less important)
Entity	(less important)
NMTOKEN or NMTokens	(less important)
A list of names:	You can specify that the value of attributes must be taken from a specific list of names (see attribute reset)

There are four kinds of default values:

#REQUIRED	The attribute must have an explicitly specified value
#IMPLIED	The attribute value is not required, and no default value is provided
#FIXED VALUE	(less important)
"Value"	An attribute can be given any legal value as default.

PIXML Tags

The root element for PIXML is the <pixml> folder tag. According to XML architecture, a folder tag is the root node of the document. There must be exactly one root node object.

Other tags are listed below:

image	single image information
border	border information.
bg	background image information.
g	group of 2 D graphics
text	text information
rect	rectangle information
circle	circle information
ellipse	ellipse information
polygon	polygon information
fill	fill area information
alpha	alpha-blending information
trans	transparency information

<pixml>

A <pixml> tag represents a folder that can contain images, graphics, and more <pixml> folders. The root node of a document must be exactly one PIXML folder tag.
The folder tag is defined as follows:

```
<!ELEMENT  pixml
   ((image | pixml)+, border?, bg*,
    (g|text|rect|circle|ellipse|line|polyline|polygon|fill)*,
    zoom?, pan?,
    event*)>
```

The following explanation may assist with understanding XML syntax:

<!ELEMENT pixml	"pixml" is the tag name.
(image | pixml)+	It must (defined by +) contain at least one image or (defined by |) folder tag.
border?	A border tag is optional (defined by ? mark).
bg*	It may have zero or more bg tags.
(g|text|rect|circle|ellipse|line|polyline|polygon|fill)*	It can have zero or more graphics tags,
zoom?, pan?	and may have optional zoom and pan tags
event*	It may have zero or more event tags

pixml attribute list:

```
<!Attlist  pixml
  id              ID                    #IMPLIED
```

url	CDATA	#IMPLIED
level	CDATA	"0"
displevel	CDATA	"6"
fcn	(zoomin, zoomout, szoomin, szoomout, pan, reset, link)	"link"
toolbar	(true, false)	"true"
menu	(true, false)	"true"
layout	(border, flow, grid, gridbag, freestyle, card)	#REQUIRED
cols	CDATA	"1"
fps	CDATA	"1"
width	CDATA	"width"
height	CDATA	"height"
depth	CDATA	"0"
x	CDATA	"0"
y	CDATA	"0"
position	(north, south, east, west, center)	"north"
dx	CDATA	"0"
dy	CDATA	"0"
hoffset	CDATA	"0"
voffset	CDATA	"0"
current	CDATA	"0"
bgcolor	CDATA	#IMPLIED
align	(LT, RT, LB, RB, center, stretch)	"center"

>

NOTE: Attributes for a child folder that are not specified are inherited from the parent. If there is no parent folder, unspecified attributes are set to the default value.

id

identifier
not required

url

specifies the <pixml> folder contents as another PIXML file

not required

If a <pixml> folder tag includes a url attribute, all tags nested inside of it are ignored. The contents of the folder will be the contents of the root folder in the reference PIXML file. This file will only be retrieved when the "level" of the folder is greater than the "displevel" of the folder. When the url is retrieved, the root folder in that file will replace the <pixml> folder with that url as an attribute.

level

image resolution level

default = 0

displevel

threshold for absolute level below which folder icon is displayed

default = 6

fcn

defines function selected when image loads

default = link

toolbar

whether or not built-in toolbar is displayed

default = true

menu

whether or not built-in 'right click' mouse menu is enabled

default = true

layout

defines the folder layout mode

required

Layout Descriptions

The layout mode attribute controls how objects are arranged within their respective containers (folders) and controls how objects may move within a container.

Border Layout

Border layout can display up to 5 objects. Each object has a geographic position: north, south, east, west and center. The cell size available for each object is determined by the order of the objects. The geographic position of items in a folder will be determined by the order in which their respective tags appear inside the <pixml> tag: The first item will be placed in the "north", the second in the "south", the third in the "east", the fourth in the "west", and the fifth in the center. In this layout, all items tags beyond the fifth item inside the <pixml> tag will be ignored.

Flow Layout

Objects are displayed one by one starting from the top left of the display window and moving to the right. When the right hand side of the window is encountered the next object is displayed on a new line below all the objects on the line above.

Grid Layout

Objects are placed in a regular grid pattern with each cell being the same size. The size of the highest and the widest objects in the array determine the cell size.

Grid Bag Layout

Similar to the Grid layout except all the cells in a row have the same vertical size as defined by the highest item in the row, and all the cells in the same column have the same size according to the widest item in the column.

Freestyle Layout

Every object can be placed anywhere in its respective container and overlaps are allowed.

Animation Loop/Card Layout

This feature is used to display a series of images in sequence over time under user control, or at a given frame rate. It can be used for applications such as a slide show or for motion sequences such as medical imagery.

specifies the number objects in each row, i.e. the number of columns in the grid

default = 1

NOTE: This attribute only has an effect when the grid or grid bag layout has been selected.

fps

specifies the number of frames per second

default = 1

NOTE: This attribute only has an effect when the animation/card layout has been selected.

width, height

specifies the width/height of display window held by this folder
default(s) = width, height

NOTE: This attribute only has an effect when the flow layout has been selected.

depth

Z-buffer depth, determines how items overlay each other

default = 0

x, y

folder position in window for freestyle layout

default(s) = 0, 0

position

folder position for border style layout

default = north

dx

horizontal distance between each object in the folder

default = 0

dy

vertical distance between each object in the folder

default = 0

hoffset

the horizontal distance between the folder origin and the virtual canvas origin

default = 0

voffset

the vertical distance between the folder origin and the virtual canvas origin

default = 0

current

used to specify which image is displayed (only for card layout)

default = 0

NOTE: This attribute only takes effect when the animation/card layout has been selected.

bgcolor

background color

not required

align

alignment of <pixml> folder within allocated space

default = center

Valid values:

center	put in center of cell
TL	put top left corner
TR	put to top right corner
BL	put to bottom left corner
BR	put to bottom right corner

stretch stretch folder to fit window

<image>

The <image> tag describes a single image. It is defined as:

```
<!ELEMENT  image
  ((border?, bg*
   (g|text|rect|circle|ellipse|line|polyline|polygon|fill)*,
   zoom?, pan?,
   link*,
   event*)>
```

The definition is very similar to the PIXML folder definition except:

- It cannot contain an image or folder tag.
- You can specify zero or more link tags with the image

image attribute list

<!AttList image

id	ID	#IMPLIED
url	CDATA	#Required
dtop	CDATA	"0"
dleft	CDATA	"0"
dbottom	CDATA	"max"
dright	CDATA	"max"
level	CDATA	"0"
bgcolor	CDATA	"white"
depth	CDATA	"0"
x	CDATA	"0"
y	CDATA	"0"
position	(north, south, east,	"north"

	west, center)	
align	(LT, RT, LB, RB, center, stretch)	"center"

>

id

identifier

not required

url

location of the image

required

If the URL is relative, the URL reference path is the folder URL.

Note that URLs can be absolute or relative, and can indicate a file or a CGI or other types of dynamic content.

dtop, dleft, dbottom, dright

define the portion of the image to display

default(s) = 0, 0, max, max (the full image)

These attributes affect the displayed portion of the <image> much in the same way that CSS rules affect clipping.

level

image resolution level

default = 0

bgcolor

background color

default = white

depth

Z-buffer depth, determines how items overlay each other

default = 0

x, y

image position in window for freestyle layout

default(s) = 0, 0

position

image position for border style layout

default = north

align

alignment of image within allocated space

default = center

Valid values:

center	put in center of cell
TL	put top left corner
TR	put to top right corner
BL	put to bottom left corner
BR	put to bottom right corner
stretch	stretch image to fit space

<bg>

<bg> defines background image information. The <alpha> and <trans> tags may applied to include specified alpha-blending and transparency information. We only support Tix background image in this version. The definition is:

<!ELEMNET bg (alpha?,trans?)>

bg attribute list

<!AttList bg

id	ID	#IMPLIED
url	CDATA	#REQUIRED

level	CDATA	"0"
dtop	CDATA	"0"
dleft	CDATA	"0"
dbottom	CDATA	"max"
dright	CDATA	"max"
align	(tile, center, LT, RT, LB, RB, stretch)	"tile"
offx	CDATA	"0"
offy	CDATA	"0"

>

id

identifier

not required

url

background image location

required

dtop, dleft, dbottom, dright

defines which portion of the image to display

default(s) = 0, 0, max, max (the full image)

These attributes affect the display of the background image much in the same way that CSS style rules affect clipping.

align

method used to position background image

default = tile

Valid values: tile --- tile from left and top

LT --- align to top left corner

RT --- align to top right corner

LB --- align to bottom left corner

RB --- align to bottom right corner

center --- display image at item center
stretch --- stretch image to whole folder window

offx, offy

horizontal and vertical offsets of background image in window

default(s) = 0, 0

Reserved value: seam --- seamlessly integrates with html background image

<border>

A border element can not include any tags. The definition is as follows:

<!ELEMENT border EMPTY>

border attribute list

<!AttList border		
id	ID	#IMPLIED
type	(empty, etched, line, bevel)	"empty"
size	CDATA	"1"
color	CDATA	"light gray"
shadow	CDATA	"dark gray"
highlight	CDATA	"white"
lx	CDATA	"0"
rx	CDATA	"0"
ty	CDATA	"0"
by	CDATA	"0"
>		

id

identifier

not required

type

border style

default = empty

size

width of border

default = 1

color

border color

default = light gray

shadow

shadow color

default = dark gray

highlight

highlight color

default = white

lx, rx, ty, by

left, right, top and bottom intervals between border and item

defaults = 0, 0, 0, 0

Border Types

1) Empty Border (no border)
   Attributes are not useful.
   Note: As item window size may be bigger than image size, so that an empty border item may look like it has a line border some times.

2) Etched Border
   size = 2 pixels

color --- border color

highlight – highlight color
Example: etched border with "grey" color and "white" highlight color.

3) Line Border
size = border line width.
color = border color

4) Bevel Border
size = border width
color = border color (if not specified, there is no outside border)
highlight = border highlight color
shadow = shadow color

<g>

A 2-D graphics tag <g> can contain one or more 2-D graphics objects. Note that all 2-D graphics object tags can have all of the attributes that a <g> tag can have. That is, <text>, <rect>, <circle>, <ellipse>, <polyline> and <polygon> tags can all have fill, fill-opacity, stoke, stroke-width, stroke-linecap, stroke-linejoin and stroke-opacity as attributes. (These attributes have been omitted from the specifications of each individual shape to avoid repetition and save space.) Any of the graphics tags will inherit these values from their parent graphics tag. If the "root" graphic tag does not specify any of these attributes, they will be set to a default.

This family of tags will produce graphics according to the same rules as the tags in the WC3 SVG specification. For instance, if a <polyline> is associated with a fill color, the <polyline> will be filled as if it were a <polygon>.

<!ELEMENT g (g|text|rect|circle|ellipse|line|polyline|polygon)+>

g attribute list

<!AttList g

id	ID	#IMPLIED
fill	CDATA	#IMPLIED
fill-opacity	CDATA	#IMPLIED
stroke	CDATA	#IMPLIED
stroke-width	CDATA	#IMPLIED
stroke-linecap	(butt, round, square)	#IMPLIED
stroke-linejoin	(miter, round, bevel)	#IMPLIED
stroke-opacity	CDATA	#IMPLIED

>

id

identifier

not required

fill

color of the shape or text (henceforth referred to as "graphic")

not required

fill-opacity

opacity of the color of the fill

not required

stroke

color of the outline of the graphic

not required

stroke-width

width of the outline in pixels

not required

stroke-linecap

describes the shape of the outline at the line endpoint

not required

stroke-linejoin

describes the shape of corners outlines that are stroked

not required

stroke-opacity

opacity of the color of the outline

not required

<text>

<!ELEMENT text EMPTY>

text attribute list

<!AttList text

id	ID	#IMPLIED
x	CDATA	"0"
y	CDATA	"0"
font-family	CDATA	"sans serif"
font-size	CDATA	"15"
font-weight	(normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900)	"normal"

>

id

identifier

not required

x

the x-coordinate for the initial text position for the text to be drawn

default = 0

y

the y-coordinate for the initial text position for the text to be drawn

default = 0

font-family

specifies the font family used to render text

default = sans serif

font-size

size of the letters (measured from baseline to baseline of text) in the current coordinate system

default = 15

font-weight

boldness/lightness of the letters

default = normal

<rect>

<!ELEMENT rect EMPTY>

rect attribute list

```
<!AttList  rect
   id              ID              #IMPLIED
   x               CDATA           "0"
   y               CDATA           "0"
   width           CDATA           #REQUIRED
   height          CDATA           #REQUIRED
   rx              CDATA           "0"
   ry              CDATA           "0"
>
```

id

identifier

not required

x

the x-axis coordinate of the side of the rectangle which has the smaller x-axis coordinate value in the current viewport's coordinate system

default = 0

y

the y-axis coordinate of the side of the rectangle which has the smaller y-axis coordinate value in the current viewport's coordinate system

default = 0

width

width of the rectangle

required

height

height of the rectangle

required

rx

the x-axis radius of the ellipse used to round off the corners of the rectangle

default = 0

ry

the y-axis radius of the ellipse used to round off the corners of the rectangle

default = 0

If a negative radius is specified, the absolute value of the radius will be used.

<circle>

<!ELEMENT circle EMPTY>

circle attribute list

```
<!AttList  circle
   id              ID              #IMPLIED
   cx              CDATA           "0"
   cy              CDATA           "0"
   r               CDATA           #REQUIRED
>
```

id

identifier

not required

cx

the x-axis coordinate of the center of the circle

default = 0

cy

the y-axis coordinate of the center of the circle

default = 0

r

the radius of the circle

required

If a negative radius is specified, the absolute value of the radius will be used.

<ellipse>

<!ELEMENT ellipse EMPTY>

ellipse attribute list

```
<!AttList  ellipse
   id              ID              #IMPLIED
   cx              CDATA           "0"
   cy              CDATA           "0"
   rx              CDATA           #REQUIRED
   ry              CDATA           #REQUIRED
>
```

id

identifier

not required

cx

the x-axis coordinate of the center of the ellipse

default = 0

cy

the y-axis coordinate of the center of the ellipse

default = 0

rx

the x-axis radius of the ellipse

required

ry

the y-axis radius of the ellipse

required

If a negative radius is specified, the absolute value of the radius will be used.

<line>

<!ELEMENT line EMPTY>

line attribute list

```
<!AttList  line
   id              ID              #IMPLIED
   x1              CDATA           "0"
   y1              CDATA           "0"
   x2              CDATA           "0"
   y2              CDATA           "0"
>
```

id

identifier

not required

x1

the x-axis coordinate of the start of the line

default = 0

y1

the y-axis coordinate of the start of the line

default = 0

x2

the x-axis coordinate of the end of the line

default = 0

y2

the y-axis coordinate of the end of the line

default = 0

<polygon>

<!ELEMENT polygon EMPTY>

polygon attribute list

```
<!AttList  polygon
   id              ID                  #IMPLIED
   points          CDATA               #REQUIRED
>
```

id

identifier

not required

points

vertices of the polygon, in the order they would appear in a path along the perimeter

required

The list of points should be inside quotation marks. Each point should consist of an x-coordinate and a y-coordinate, separated by a comma. The points themselves should be separated by whitespace.

Example:

points="100,200 300,400 500,200" (This set of points describes a triangle.)

When the polygon is rendered, there will be an edge between adjacent points in the list and between the first and last points as well.

<polyline>

<!ELEMENT polyline EMPTY>

polyline attribute list

```
<!AttList  polyline
   id              ID                  #IMPLIED
   points          CDATA               #REQUIRED
>
```

id

identifier

not required

points

endpoints of the line segments that make up the polyline, in the order they would appear traversing the polyline end to end

required

<alpha>

The <alpha> tag defines alpha-blending information between its parent and its parent's image/folder.

<!ELEMENT alpha EMPTY>

alpha attribute list

<!AttList alpha

id	ID	#IMPLIED
value	CDATA	"0"
range	(all, nonsvg, individual)	"image"

>

id

identifier

not required

value

alpha blending value (0~65536)

default = 0

pixel_value = (this object) * **value/65536 + (1-value/65536)***(other object)

range

the item, the image but not any of the graphics defined within the item, or the item and its subtags

default = image

<trans>

The <trans> tag defines alpha-blending information between its parent and its parent's image/folder. The element and attribute list definitions are:

<!ELEMEMT trans EMPTY>

trans attribute list

<!AttList Trans

id	ID	#IMPLIED
key	CDATA	"black"
range	(all, nonsvg, individual)	"individual"
back	(this,other)	"this"

>

id

identifier

not required

key

used to define which color will be transparent

default = black

range

the item, the image but not any of the graphics defined within the item, or the item and its subtags

default = individual

back

declares which object is associated with transparency color

default = this

<link>

<!ELEMENT link EMPTY>

link attribute list

<!AttList Link

id	ID	#IMPLIED
href	CDATA	#REQUIRED
target	CDATA	"new"
ltext	CDATA	#IMPLIED
atop	CDATA	"0"
aleft	CDATA	"0"
abottom	CDATA	"max"
aright	CDATA	"max"

>

href

the url to be accessed

required

If the URL is relative, its reference URL is the URL of the associated image.

target

target window

required

Reserved values: new --- new window

self --- same window

parent --- parent window

notoolbar --- window with no toolbar

ltext

specifies the text associated with the hyperlink

default = the whole <text> sentence (if the <link> is associated with a <text> object)

atop, aleft, abottom, aright

specifies over what part of the image the link is active

default(s) = 0, 0, max, max (the entire image)

These attributes affect the active area of the link much in the same way that CSS rules affect clipping.

<in>/<out>/<ino>

Display behavior tags specify how the items zoom and pan. In PIXML, we have two major objects, images and folders. Naturally, they can have their own zoom behaviors. For more flexibility, we may specify each item's zoom behavior. If the behavior is not specified, the item inherits a behavior from its parent or default value.

The most important part of a zoom is the zoom center, which identifies the center when a window is zoomed to different levels. There are two tags working for the center definition.

The <in> tag applies when the entire item is displayed within the viewing window. The <out> tag applies when the entire item is not displayed within the viewing window. The <ino> tag allows the programmer to specify the same rule for both of these cases at one time.

<!ELEMENT in subw>

<!ELEMENT out subw>

<!ELEMENT ino (subw*)>

in/out/ino attribute list

```
<!ATTList in
  id              ID          #IMPLIED
  zoomlevel       CDATA       "0"
  inside          CDATA       #IMPLIED
  outside         CDATA       #IMPLIED
  both            CDATA       #IMPLIED
>
<!ATTList out
  id              ID          #IMPLIED
  zoomlevel       CDATA       "0"
  inside          CDATA       #IMPLIED
  outside         CDATA       #IMPLIED
  both            CDATA       #IMPLIED
>
<!ATTList ino
```

```
  id            ID          #IMPLIED
  zoomlevel     CDATA       "0"
  inside        CDATA       #IMPLIED
  outside       CDATA       #IMPLIED
  both          CDATA       #IMPLIED
>
```

id

identifier

not required

zoomlevel

relative image level

default = current level (0)

If level = 0, consider the image in the current level, otherwise, use level + n to judge the image level.

inside

zoom center point behavior when mouse pointer is inside the item window on the zoom

not required

outside

zoom center point behavior when the mouse pointer is outside the item window on the zoom

not required

both

zoom center point behavior for both windows

not required

Center behavior values:

DC_C Keep display center not changed
IC_C Move center of image to center of display window
P_C Move refer pointer (usually mouse click point) to center of window
P_N Let refer pointer at the same position in display window
PX_xx_PY_yy xx,yy =number, percentage of move from reference pointer to display window center.
FIT This Item fit the display window.

<zoom>

<!ELEMENT zoom (((in?, out?)* | ino?))>

zoom attribute list

<!AttList zoom

id	ID	#IMPLIED
ratio	CDATA	"2"
mode	CDATA	"recenter"
max	CDATA	"0"
min	CDATA	"16"
range	(individual, nonsvg, all)	"all"
rearrange	(true, false)	"true"

>

id

identifier

not required

ratio

magnification of zoom

default = 2

mode

method used to specify placement of objects after a zoom

default = recenter (the mouse position at the zoom becomes the display center)

max

maximum display level

default = 0

For an image, this attribute defines the maximum value for the folder level + the image level.

min

minimum display level

default = 16

If the image size < 64x64 pixels, the image is not displayed

range

defines which items will be zoomed: the item in which the zoom tag is located, the item in which the zoom tag is located but not any of the graphics defined within the item, or all items

default = all

rearrange

specifies whether or not items in range are rearranged according layout rules after the zoom

default = true

<pan>

The <pan> tag defines the behavior of panning when you pan outside the window.

<!ELEMENT pan EMPTY>

pan attribute list

<! Attlist pan

id	ID	#IMPLIED
hscroll	(loop, none, stop)	"none"
vscroll	(loop, none, stop)	"none"
win	CDATA	"display"
range	(individual, nonsvg, all)	"all"
rearrange	(true, false)	"true"

>

id

identifier

not required

hscroll

specifies the pan behavior in the horizontal direction, not required

default = none

Valid values: none --- no specified behavior
stop --- item cannot go beyond window
loop --- if item is goes outside the window, it is wrapped around

vscroll

specifies the pan behavior in the vertical direction, not required

default = none

range

defines which items will be panned: the item in which the pan tag is located, the item in which the pan tag is located but not any of the graphics defined within the item, or all items

default = all

rearrange

specifies whether or not items in range are rearranged according layout rules after the pan

default = true

<event>

The <event> tag represents a type of callback function within an ordinary browser. It is used to determine which events will be handled by externally defined functions (such as in scripts.)

<!ELMENT event EMPTY>

event attribute list

<AttList event

mousemove	CDATA	#IMPLIED
mouseover	CDATA	#IMPLIED
mouseout	CDATA	#IMPLIED
lbuttonup	CDATA	#IMPLIED
lbuttondown	CDATA	#IMPLIED
rbuttonup	CDATA	#IMPLIED
rbuttondown	CDATA	#IMPLIED
click	CDATA	#IMPLIED
dblclick	CDATA	#IMPLIED
load	CDATA	#IMPLIED
keydown	CDATA	#IMPLIED
keyup	CDATA	#IMPLIED
begindraw	CDATA	#IMPLIED
enddraw	CDATA	#IMPLIED
resize	CDATA	#IMPLIED

>

All event attributes are internally controlled by default. Unless included in the <event> tag, events will not be visible to external control mechanisms.

Event	Description
mousemove	mouse moves
mouseover	mouse moves over this item
mouseout	mouse moves off this item
lbuttonup	mouse left button release
lbuttondown	mouse left button pressed
rbuttonup	mouse left button released
rbuttondown	mouse left button pressed
click	mouse click
dblclick	mouse double click
load	PIXML begins to load from server
keydown	keyboard pressed
keyup	keyboard released
begindraw	this item will be drawn
enddraw	drawing finished
resize	resize item

The following steps attach an event handler to an event:

[001] In the <event> tag, indicate which function will respond to some event (such as mouse click, key press, etc...)

The following code will cause the plugin to look for an externally defined event handler only when the mouse is double-clicked or the item (inside which the event tag is found) is resized.

<event dblclick=true resize=true>

[002] Define a JavaScript or VBScript function using the following naming convention:

Suppose the <event> tag is intended to apply to an item with an id, call it *itemid.* The event handler should be named *(itemid)*_on*(eventname)*, where *eventname* is the type of event. For instance, for an image with id *myimage*, the event handler controlling the response to a double-click of the mouse would be a function named "myimage_ondblclick".

When the mouse is double-clicked over *myimage*, the function "myimage_ondblclick" will be called. (A possible feature for later versions of PIXML is the ability to specify arbitrary names for event handlers.)

Application of Layout Rules

A layout style determines how PIXML content is organized on the PIXML canvas. First, the sizes of the all objects are calculated. In the case of single images, the image size is calculated using the image resolution and the absolute image level. To this is added any additional size occupied by associated graphics elements, borders, etc. The following steps determine the layout to be displayed:

[003] Recall the size of a folder object is defined in PIXML or inherited from a webpage.

[004] Calculate each object size at the current level.

If it is an image:

size = (image size) divided by 2^(absolute image level)

If it is a folder:

if (absolute folder level > "displevel" of folder)

size = folder icon size

otherwise

size = (folder size) divided by 2^(absolute image level)

[005] Modify each object's size by adding to it the contributions from its associated graphics and border objects. For example, if there is a text title right below the image, the size of the text region will be added in to actual object size.

[006] Apply the layout design rules with the objects' sizes. For example, in a grid layout, each cell size is equal to the maximum size of all the objects in the folder.

[007] Organize all the objects together according to their layout.

[008] We get the rectangular region for each object in the PIXML canvas coordinate system.

[009] Calculate the actual rectangular region for each item according to the rectangular region, which we get in step d), as well as its alignment.

2-D Graphic Object Extension

PIXML will have a mechanism to allow the 2-D graphic capabilities to be extended.

This will most likely be through a dll-based library (This mechanism does not work for Java). The way we deal with this type of extension will be as follows:

[0010] Within PIXML, we define a new tag "extg".

[0011] With the extg attribute, you will be able to include an additional script within PIXML and specify a dll name.

[0012] PIXML plugins open this dll and call a render function to retrieve the memory bitmap for this graphics object.

[0013] PIXML renders this object with other images with alpha-blending and transparent element

Resolution Level

PicSurf can describe the resolution of an image or a folder by the image level or by a percentage of the original. Level 0 is the full-sized original image; level 1 is half the size in both axes, and so forth.

When an object is displayed, the level of the object is combined with the level of the parent folder, and the levels of all the parent folders to compute an 'absolute level' for display purposes.

In general, "zoomin" will reduce the level of the root folder by 1 and zoomout will increase the root folder level by 1.

Zoomin and zoomout functions can also be defined by attribute values that can be integer resolution levels, percentages, or 'fit', where the zoom operation will match the resolution of the object(s) to the parent folder.

A "displevel" attribute is used to determine whether a folder is displayed as a folder icon or whether the contents of the folder are displayed. If the folder absolute level is greater than "displevel", the folder is displayed as an icon. Otherwise, the viewer displays the folder's contents.

Canvas and Viewport

For all media, the PIXML canvas is defined as the space where the PIXML content is rendered. The PIXML content includes any displayable content in PIXML such as images, folders, 2-D graphic objects, borders, etc. The canvas is infinite for each dimension of the space. However, rendering occurs inside a finite rectangular region called the PIXML viewport.

The pixel viewport is the viewing area where the PIXML content is displayed. The top-level viewport is usually the plugin display window. Each folder also has its own viewport.

The size of the root node viewport is determined by negotiation process between the PIXML document viewer and its parent (e.g., web browser).

Coordinate System

Both the PIXML canvas and PIXML viewport have their own coordinates and origin. The default unit is the pixel. However, in some cases, alternate units could be added in the future.

Positioning

The PIXML viewport is mapped to the PIXML canvas using an offset representing the x,y distance between the canvas origin and the viewport origin expressed in pixels. Each PIXML folder has its own viewport and its own virtual canvas. Each folder has a pair of offset attributes to represent these values. During a 'pan' operation, the offset values are changed according to the panning mode being used.

freestyle positioning

In 'freestyle' layout mode, objects may be mapped to the virtual canvas using absolute pixel coordinates, or may be defined as relative to another object. In this case, relative coordinate values are used.

Inheritance

In general, unspecified root folder attributes are set to their default values, whereas sub-folders inherit unspecified attributes from their respective parent folders.

<pixml> dimensions

When the width and height of a sub-folder are unspecified, the sub-folder inherits the width and height of its parent folder, adjusted according to the relative difference in their levels. For example, if a 512x1024 pixel parent folder, displayed at level 0, contains a sub-folder with unspecified dimensions, displayed at level 2, the width of the sub-folder will be 128 pixels and the height will be 256 pixels.

If the root folder has unspecified dimensions, it will inherit the width and height from the plugin display window (as they are defined in the web page that uses the PIXML file with that root folder).

bg versus bgcolor

If a folder has a background image defined by a single <bg> tag (or even several <bg> tags), that background will serve as the background for all the items that the folder contains, regardless of whether or not the items themselves contain <bg> tags or specify a bgcolor.

However, if a folder has no background image, the background for the items within the folder will be determined by their bgcolor attributes. If any sub-folder doesn't have a bgcolor attribute, it will inherit the bgcolor from the folder. Note that the root <pixml> folder cannot inherit a bgcolor value because it has no parent folder. Thus, the bgcolor of a root <pixml> folder lacking a bgcolor attribute will be set to the default color, white. Also, if any image in the folder doesn't have a bgcolor attribute, its bgcolor will also be set to the default (white).

<zoom> behavior

If the zoom behavior for an item is unspecified, it will inherit the behavior from its parent. This mode of inheritance promotes the following two results:

1) If the zoom behavior of the parent tag is changed dynamically, that change will affect the zoom behavior of all the child tags.
2) If the zoom behavior of any of the child tags is changed, that change will apply to all the child tags as well as the parent tag.

Note, however, that this type of inheritance does not extend across .pixml file boundaries. That is, if a<pixml> tag references another file with its "url" attribute, the pixml objects in the new file will not inherit any zoom behavior from the original file. If the root folder in a .pixml file has no <zoom> tags, its zoom behavior will be set to the default.

url and viewing window

Consider a <pixml> folder whose contents are found in an url reference in the <pixml> tag. The dimensions of the viewing window through which the contents of the folder will be viewed are determined by the properties of the original folder, not the properties of the new root folder in the PIXML file referenced by the url. This rule will determine the folder's viewing window dimensions, regardless of whether or not the "level" of this folder is above the "displevel" threshold and the contents of this folder are initially retrieved and displayed.

shared graphics attributes

The fill, fill-opacity, stroke, stroke-width, stroke-linecap, stroke-linejoin and stroke-opacity attributes are attributes shared by all of the graphics tags (<g> tags and those that can be contained therein.) If any of these tags do not specify any of these attributes, the attribute values will be inherited from their parent <g> tag. If the "root" <g> tag, one that has an <image> or a <pixml> tag as a root, doesn't specify any of these attributes, they will be set to the following default values:

attribute	default
fill	none
fill-opacity	1
stroke	none
stroke-width	1
stroke-linecap	butt
stroke-linejoin	miter
stroke-opacity	1

Reusability and Scripting

PIXML is object-based reusable. Each element is the object in PIXML, it refer by its id attributes. The basic requirement of id attributes is that an id name only can be defined within a PIXML document once. All id names within the document must be unique.

What will PIXML look like?

*** XML file structure

[0014] First, <?XML ....?> . This XML markup declaration to explicitly identifies the document as an XML document and indicated the version of XML to which it was authored.

[0015] Document type declarations

[0016] <root tag>

[0017] </root tag>

Comment is the same as
*** End of XML file structure

Here is the sample of PIXML

```
<?xml version="1.0"?>
<!DOCTYPE PIXML SYSTEM "
<PIXML id=root fcn=zoomin bgcolor=red style=grid>
    <image id=img1 src=http://image.picsurf.com/aa.tix level=3>
    <image id=img2 src=http://image.picsurf.com/bb.tix level=1>
    <event mousedown=true>
</PIXML>
```

The above defines a PIXML file with two images. PIXML also responds to the event "function change".

Reusability

As per the example above, you can use:
"Img1.Level = 4" to make the image 2x2 smaller (next level)
"Img2.URL=www.web4view.com/mm.tix" to change to another image.

A plugin or applet will have two parameters, SRC and CHG:
SRC --- specifies the PIXML file location
CHG --- specifies the change such as "img1.Level=4"

With this mechanism, similar applications can use the same reusable PIXML file. It also enables the server to create dynamic html pages.

Scripting

PIXML has some scripting functionality. It works similarly to reusability. Reusability worked when plugin or applet initialized just like different parameters, while scripting worked after plugin or applet is start.

For the same example:

1) You can run the "change script" when plugin is running. The plugin can accept the change and adjust its content or behavior dynamically

2) You might have several radio buttons (to control plugins)

However, you may also gain the control through right-button menu or built-in buttons. To solve consistency problems, you need to write the browser-based scripting functions using JavaScript or VBScript as follows:

Suppose, for example, the root <pixml> folder id is "root" and the mouseover event handler "root_onmouseover" is written in JavaScript or VBScript. Within root_onmouseover, it is possible to change PIXML object properties.

Tracking Events

To take full advantage of PIXML reusability and scripting capabilities, PIXML provides access to information about events triggered by the user. In the JavaScript event model, it is possible to access information generated at the time of the event, such as the position of the mouse (in several coordinate systems), or the object over which the mouse was hovering. In the sam0e way, the author of a script can obtain data from events within the PIXML plugin window.

Information associated with events can be passed to event handler functions. By default, no parameters are passed to event handlers, but the author can choose to pass them in the definitions of event handlers. The following defines the interface for passing parameters that must be used if the author chooses to pass parameters to the event handler.

***id*_onmousemove(mouseX, mouseY, dx, dy, eventId)**

***id*_onmouseover(mouseX, mouseY, eventId)**

***id*_onmouseout(mouseX, mouseY, eventId)**

***id*_onlbuttondown(mouseX, mouseY, eventId)**

***id*_onlbuttonup(mouseX, mouseY, eventId)**

***id*_onrbuttondown(mouseX, mouseY, eventId)**

***id*_onrbuttonup(mouseX, mouseY, eventId)**

***id*_onclick(mouseX, mouseY, eventId)**

***id*_ondblclick(mouseX, mouseY, eventId)**

***id*_onload(eventId)**

***id*_onkeydown(key, eventId)**

***id*_onkeyup(key, eventId)**

***id*_onbegindraw(eventId)**

***id*_onenddraw(eventId)**

***id*_onresize(sizeX, sizeY, resizeX, resizeY, eventId)**

mouseX,mouseY
: the x and y coordinates of the mouse pointer at the time of the event, in the coordinate system of the item that contains the <event> tag, at the 0-level scale of the item

dx,dy
: the changes in coordinates of the mouse pointer in the mousemove event

key
: the value of the key, pressed or released

sizeX, sizeY
: the dimensions of the item before being resized

resizeX, resizeY
: the dimensions of the item after being resized

eventId
: the id of the <item> tag directly inside which the <event> tag is found

This JavaScript example illustrates the implementation of two event handlers, one that gets passed parameters, and another that does not:

```
<SCRIPT language="JavaScript">

function myitem_onmouseover(mouseX, mouseY, eventId) {

    if( mouseX == 0)
        alert("mouse moved over myitem from left!")
```

```
        else if  (mouseX == eventId.width)
                alert("mouse moved over myitem from right!")

}

function myitem_onmouseoff() {

        alert("mouse moved off of my item!")

}

</SCRIPT>
```